CAMPUS
DE EXCELENCIA
INTERNACIONAL

Graduado en Ingeniería Informática

Universidad Politécnica de Madrid

Facultad de Informática

TRABAJO FIN DE GRADO

Visualizador de programas Java Concurrentes

Autor: Víctor de la Peña Martinez

Director: Julio Mariño

MADRID, JUNIO DE 2014

TABLA DE CONTENIDO

1. RESUMEN

1.1. EN ESPAÑOL.

El presente trabajo fin de Grado, que, a partir de ahora, denominaré TFG, consiste en elaborar una monitorización de programas concurrentes en lenguaje Java, para que se visualicen los eventos ocurridos durante la ejecución de los dichos programas. Este trabajo surge en el marco de la asignatura "Concurrencia" de la Escuela Técnica Superior de Ingeniería Informática de la Universidad Politécnica de Madrid, impartida por D. Julio Mariño y D. Ángel Herranz.

El objetivo principal de este proyecto es crear una herramienta para el aprendizaje de la asignatura de concurrencia, facilitando la comprensión de los conceptos teóricos, de modo que puedan corregir los posibles errores que haya en sus practicas.

En este proyecto se expone el desarrollo de una librería de visualización de programas concurrentes programados en Java, usando un formalismo gráfico similar al empleado en la asignatura. Además esta librería da soporte a los mecanismos de sincronización usados en las prácticas de la asignatura: la librería Monitor (desarrollada por los profesores de la asignatura, D. Ángel Herranz y D. Julio Mariño) y la librería JCSP (Universidad de Kent).

1.2. EN INGLÉS.

This Bachelor Thesis addresses the problem of monitoring a Java program in order to trace and visualize a certain set of events produced during the execution of concurrent Java programs. This work originates in the subject "Concurrency" of the Computer Science and Engineering degree of our University.

The main goal of this work is to have a tool that helps students learning the subject, so they can better understand the core concepts and correct common mistakes in the course practical work.

We have implemented a library for visualizing concurrent Java programs using a graphical notation similar to the one used in class, which supports the design of concurrent programs whose synchronization mechanisms are either monitors (using the Monitor package) or CSP (as implemented in the JCSP library from Kent University).

2. INTRODUCCIÓN Y OBJETIVOS.

El proyecto surgió a partir de la dificultad que se presenta a los alumnos al tener que visualizar la ejecución de un programa con varios procesos concurrentes, en la asignatura "Concurrencia" del Grado en “Ingeniería Informática” y del Grado de “Ingeniería Informática y Matemáticas”.

En la actualidad, los alumnos disponen de librerías de entrada/salida concurrente que facilitan la impresión de trazas, tanto al realizarse en una ventana para todos los procesos, como con una ventana por cada proceso. Estas librerías son, además, de una cierta ayuda a la hora de localizar errores en el código que se desarrolla.

No obstante, la experiencia de los docentes, tanto en el aula como, especialmente, en las tutorías, demuestra que el uso de una notación gráfica para representar las interacciones entre los procesos suele ser mucho más útil a la hora de visualizar el comportamiento de los programas concurrentes.

Por este motivo y con la idea de facilitar el aprendizaje de la asignatura, se ha considerado la necesidad de representar gráficamente las interacciones entre los procesos, creando una librería de visualización de programas concurrentes programados en Java, usando un formalismo gráfico similar al empleado durante la exposición de la asignatura.

Esta nueva librería sirve como soporte a los mecanismos de sincronización usados en las prácticas de la asignatura, véase las librerías Monitor y JCSP, desarrolladas por D. Ángel Herranz y D. Julio Mariño y por la Universidad de Kent, respectivamente.

En la siguiente tabla se exponen los objetivos que se fijaron a principio de semestre para la realización del TFG.

Objetivo 1	Análisis de Requisitos del problema.
Objetivo 2	Exploración de librerías de visualización gráfica, con vistas a su posible uso en el desarrollo.
Objetivo 3	Desarrollo del visualizador de monitors&conditions.
Objetivo 4	Desarrollo del visualizador de canales síncronos con recepción alternativa condicional (JCSP).
Objetivo 5	Pruebas de la solución con ejemplos de la asignatura.

3. ANÁLISIS DEL PROBLEMA

A continuación se va a plantear un ejemplo donde se verá claramente los problemas que surgen y de los que ha derivado la realización de este TFG.

Se explicará y simulará la visualización de un programa concurrente de Lectores-Escritores donde ambos escriben en el recurso compartido, que será un buffer.

En este ejemplo, se usará el código de Monitor. Este código presenta unos métodos para evitar las condiciones de carrera en el recurso compartido, y no permitir que haya un escritor y un lector a la vez, ejecutando dentro de la sección critica, donde está el recurso compartido.

Con la ejecución de un programa con Lectores-Escritores y un recurso Compartido, el código elaborado en este TFG, generará un fichero capturando los eventos que nos interesan. Los eventos a capturar son: la creación de un monitor, la creación de una condition, al entrar al método await() de Monitor.java , al salir del método await() de Monitor, al entrar y al salir del método signal() de Monitor.java.
El esquema de los datos que componen un evento seria:

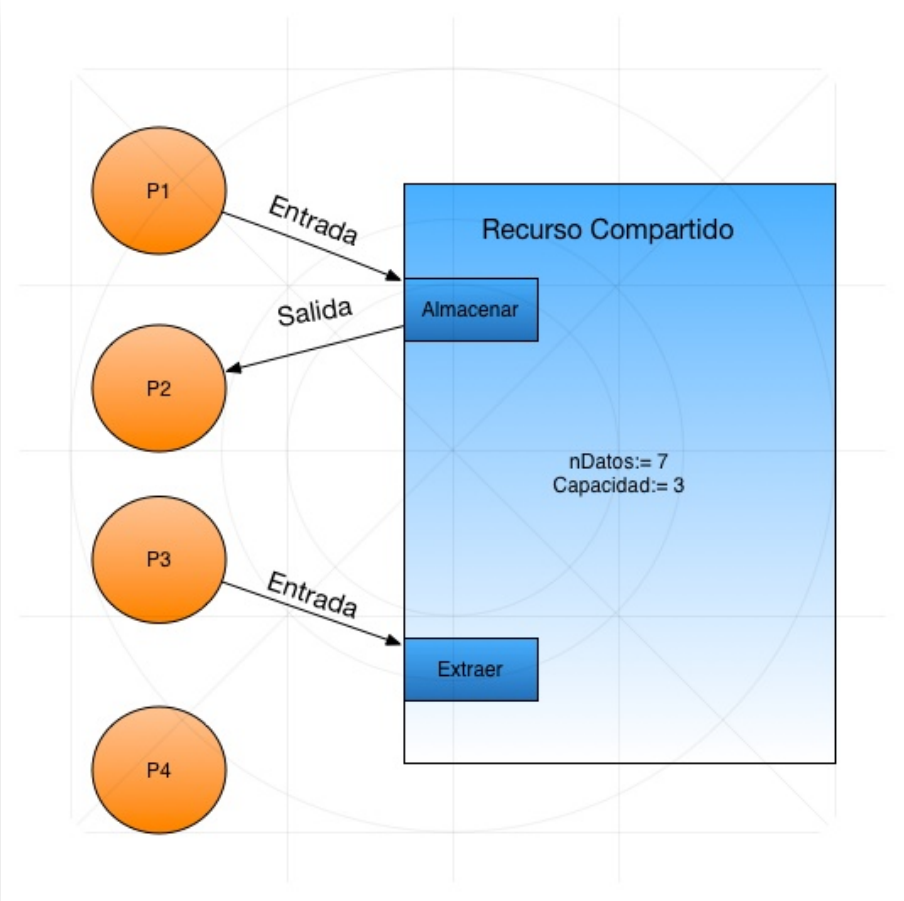
```
[Delimitador de evento]
[Tipo de Evento]
[Proceso que realiza el evento]
[Fecha en que sucede el evento]
[Variables de interés del Recurso Compartido]
[Delimitador de Evento]
```

Unas líneas de este fichero de instrumentalización o monitorización serian así:

```
============================
“P1”            //P1 Es un proceso escritor
“entrar_await()”
“Variables Recurso Compartido:
          nLectores:=2
          nEscritores:=0”
“2014-04-18 11:17:07.992”
============================
“P2”            //P2 Es un proceso lector
“salida_await()”
“Variables Recurso Compartido:
          nLectores:=1
          nEscritores:=0”
“2014-04-18 11:17:25.854”
============================
```

```
“P3”            //P3 Es un proceso lector
“entrada_signal()”
“Variables Recurso Compartido:
          nLectores:=0
          nEscritores:=0”
“2014-04-18 11:17:35.102”
============================
“P4”            //P4 Es un proceso escritor
“salida_await()”
“Variables Recurso Compartido:
          nLectores:=0
          nEscritores:=1”
“2014-04-18 11:17:39.232”
============================
```

Una vez elaborado el fichero de eventos, se ejecutaría el simulador con el fichero de eventos como entrada y en él se mostraría la visualización de dichos eventos. A continuación, se presenta una captura de pantalla de cómo podría ser la visualización de los eventos, en este caso se simula el primer evento:


P1
Entrada
Recurso Compartido
Salida
Almacenar
P2
nDatos:= 7
Capacidad:= 3
P3
Entrada
Extraer
P4

4. DISEÑO DE LA SOLUCIÓN

Para un mayor entendimiento del diseño de este TFG, se va a dividir en dos partes: gestor de eventos y captura de dichos eventos y visualizador de eventos.

4.1. GESTOR DE EVENTOS Y CAPTURA DE DICHOS EVENTOS.

Inicialmente se llevó a cabo un estudio del modo de realizar la instrumentalización o monitorización del código para capturar eventos de los programas concurrentes.

En la primera iteración del estudio de dicho problema, se valoró la realización con librerías de java de monitorización, pero se desechó debido a que monitorizaban la maquina virtual pero no el código, que era lo se que quería lograr.

Se optó por realizar el código de monitorización dentro del código de la clase Monitor.java, proporcionada por los profesores de la asignatura.

Esta última opción se ha ido refinando y ha permitido realizar un estudio sobre los datos resulta interesante recoger cuando ocurre un evento. En este proceso han ido surgiendo una serie de problemas, como la forma de coger datos de una clase que llama a un objeto Monitor. Los datos que se van a capturar en cada momento de lanzar un evento van a ser:

- Tipo de evento
- Proceso que realiza el evento.
- Fecha en que sucede el evento.
- Variables del recurso compartido.

En la recopilación de estos datos, la que ha dado mayor problema es la captura de las variables del recurso compartido, ya que por limitaciones del lenguaje Java, se han tenido que depurar las opciones que nos da el lenguaje hasta llegar a la solución.

Esta solución pasa por las características de herencia, para que podamos recoger las variables del recurso compartido. Esto significa que, para recopilar las variables del recurso compartido, esta clase hereda de la clase Monitor.java y así en tiempo de ejecución el código, que se escribe en Monitor, pueda tener acceso a las variables del recurso compartido, y así se pueda generar el evento con la información que se quiere recoger para su simulación.

Para la realización de este apartado se diseñan varias clases contenedoras de información, así como definiciones de interfaces.

Las clases auxiliares e interfaces que se incluirán en la librería facilitada por los profesores de la asignatura de Concurrencia, cclib, para que se pueda realizar la captura de eventos, son las siguientes:

- Evento.java
- Transparente.java
- EventManager.java
- EventoImpl.java
- Par.java

Evento.java: Es un interfaz con los métodos y operaciones que se han diseñado para el evento y sus atributos.[Ver código Evento.java]

EventoImpl.java: Es la clase que implementa los métodos descritos en el interfaz de Evento.java.[Ver código EventoImpl.java]

Transparente.java: Es el interfaz que define los métodos y operaciones para el recurso compartido, es decir, el recurso compartido que se usa tiene que implementar Transparente. [Ver código Transparente.java]

EventManager.java: Es la clase que realiza la gestión de eventos, abriendo el fichero donde se escribirán los eventos, y los irá recogiendo según vayan ocurriendo. [Ver código EventManager.java]

Par.java: Clase contenedora que recogerá los valores que nos interesen del recurso compartido, con la dupla nombre de la variable, y el valor de la misma.[Ver código Par.java]

Una vez desarrolladas estas clases, y codificado un ejemplo, este apartado generara un fichero de texto con este formato:

```
===============<numero de Evento>================
nombreProceso:
<numero hash del proceso>
Operacion:
<Entrada o Salida Await/Signal>
Variables Recurso Compartido:
Operación: <Operación del recurso compartido>
<Variables de interés del recurso compartido>
<Nombre de variable>: <Valor>
++++++++++++++
TimeStamp:
<Time Stamp>
===============<numero de Evento>================
```

Un ejemplo de eventos capturados, sería:

```
===============0===============
nombreProceso:
-1030971579
Operacion:
Entrada signal
Variables Recurso Compartido:
Operación: Extraer
capacidad: 6
nDatos: 2
++++++++++++++
TimeStamp:
2014-05-26 17:26:28.452
===============0===============
...
===============53===============
nombreProceso:
-1030971579
Operacion:
salida signal
Variables Recurso Compartido:
Operación: Extraer
capacidad: 6
nDatos: 10
++++++++++++++
TimeStamp:
2014-05-26 17:26:28.516
===============53===============
```

Tras la obtención de un fichero con una lista de eventos, con la información considerada importante, se decide pasar a la codificación y proyección del visualizador.

4.2. VISUALIZADOR DE EVENTOS.

En la elaboración de este apartado del proyecto se ha necesitado la creación de otro programa aparte del de la monitorización, aunque usa la librería común cclib1.

La principal complejidad que se encuentra en este apartado del proyecto, es la del dibujo de los eventos y la representación de cada uno en un instante de tiempo determinado.

Para conseguir la visualización del proyecto se han creado las siguientes clases:

- Values.java
- Simulate.java
- Visual.java
- MainModel.java

Values.java: Clase contenedora de coordenadas para pintar las figuras geométricas .[Ver código Values.java]

Simulate.java: Esta clase, al ser creada desde MainModel.java, con la lista de eventos y la lista de procesos existentes, tiene un metodo simulate(), que crea las estructuras de datos necesarias de la lista de eventos (de lista de eventos a lista de objetos de tipo Values) y de procesos a una lista de objetos tipo Values para, al crear el objeto Visual y pasárselos como atributos de la clase, pueda construir dichas estructuras y pintarlas.

Además es la responsable de gestionar la escala de tiempos para la simulación, por lo que es la clase que más peso tiene de este apartado. Se ha creído oportuno que fuera así, para que el proceso de pintado se refrescara de manera independiente del tiempo de espera que hay entre eventos, ya que en esta primera versión hemos optado por la opción de realizar un intervalo de tiempo estándar entre eventos. [Ver código Simulate.java]

Visual.java: Es la responsable de pintar todas las figuras geométricas del proceso, del recurso compartido y de los eventos. Ésta depende de los valores pasados como atributos para que pueda generar las estructuras necesarias y crear un JFrame donde poder pintar la simulación.

Esta clase extiende la clase JPanel, esto facilita que se pueda redefinir los métodos de pintar paint(Graphics g), y paintComponent(Graphics g). Éstas llaman a métodos auxiliares, creados para este proyecto, que realizan el pintado y manejo de los eventos, así como a las figuras geométricas que representan a los procesos, y al recurso compartido.[Ver código Visual.java]

MainModel.java: Es la clase principal para hacer funcionar el simulador. Es la encargada de leer el fichero de eventos y crear la estructura de datos para poder emularlo gráficamente, pasándoselo al objeto Simulate, creado de la clase Simulate.java.

La mayor complejidad observada en la elaboración de esta clase es la lectura y comprensión, por parte del código, de cuándo termina y empieza cada apartado reseñable del evento. Esta última parte ha sido la que mayor atención se ha llevado, más que por su complejidad en cuanto a cómo resolverse, por el tiempo que ha llevado depurar la forma de conversión de texto a las distintas clases de los atributos de la clase Evento, en las que la conversión ha sufrido varios procesos de depurado para su correcta lectura y comprensión . [Ver código MainModel.java]

Una vez definidas las clases que componen este apartado del proyecto y, tras un proceso de refinamiento del código, para que se visualice finalmente los procesos y el recurso compartido, comienzo a realizar un test con un ejemplo de proceso concurrente que detallo en el siguiente apartado.

5. RESULTADOS

Se ha usado el ejemplo Lectores/Escritores que escriben en un recurso compartido para comprobar la ejecución del proyecto elaborado. Para realizar este ejemplo, se diseñan dos clases que escribirán o leerán del recurso compartido. Estas son las clases Escritor.java *y* Lector.java. Dichas clases extienden la clase Thread.

La clase *Escritor*, es la encargada de escribir en el recurso compartido 'A', 'B','C','D','E'. Con un tiempo de espera de 5 milisegundos entre diferentes intentos de escrituras.

En cambio la clase *Lector*, es la encargada de leer desde el recurso compartido, con un retardo entre una lectura y la siguiente.

El recurso compartido (Clase RecursoCompartido.java) tiene que extender a la clase Monitor.java e implementar el Interfaz Transparente.java. Con esto se asegura el acceso a los recursos de monitor, y a la implementación del método showMe(). Método en el cual se decide qué variables del recurso compartido son de interés para mostrarse en la simulación de los eventos creados durante los programas concurrentes.

La clase Main.java es la encargada de crear los objetos:

- **EventManager**, pasándole, si se le ha pasado, el nombre del fichero, sino EventManager creará uno por defecto llamado eventos en la carpeta del código que lo este ejecutando.

- **Recurso compartido.**

- **Lectores y Escritores,** que son los encargados de escribir y leer del recurso compartido.

Una vez definidos los objetos, comenzara la creación de hilos lector e hilos escritor, esperando por ellos hasta que terminen su ejecución.

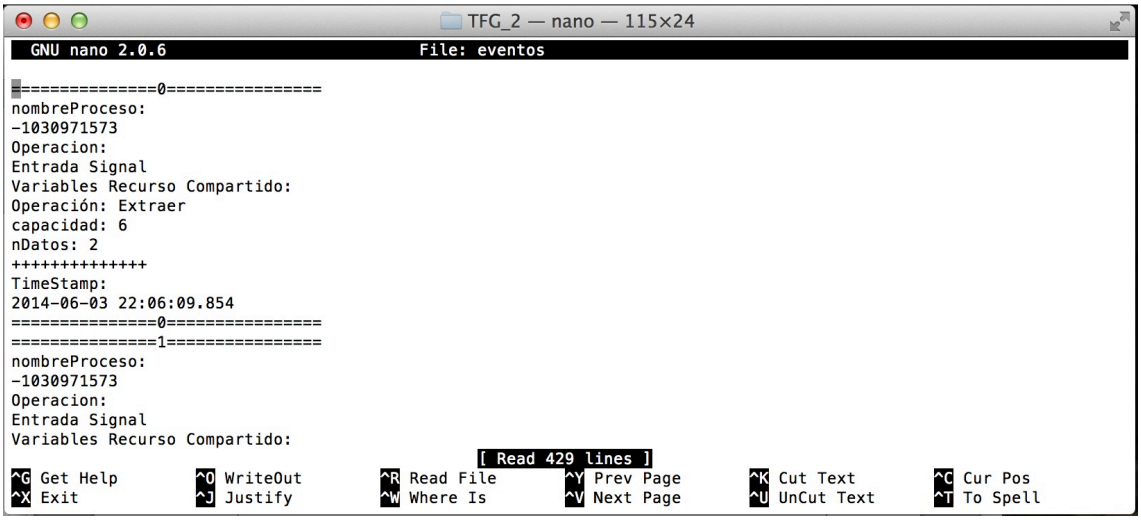
Una vez ideado el ejemplo, se procede a realizar la demostración del proyecto.

El primer paso es generar el fichero con los eventos. Se muestra este proceso en las siguientes capturas de pantalla, donde se ve la llamada al programa que se utiliza como ejemplo y cómo comprobar si el contenido del fichero creado es correcto.

Para ello usare que me lo guarde en el fichero por defecto eventos.

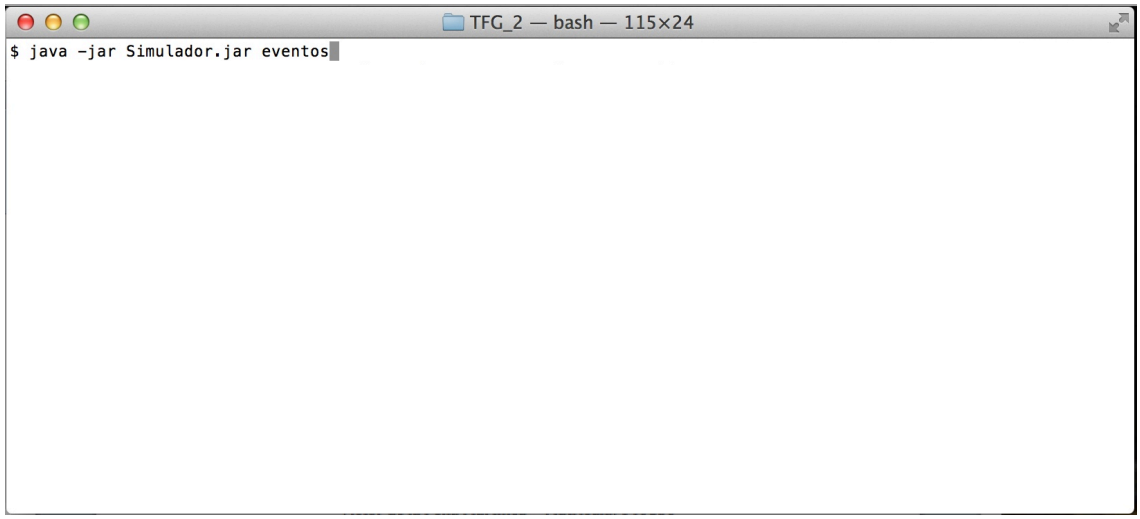
1. Invocación del ejecutable *Ejemplo Lectores_Escritores.jar*

2. Comprobación de que el fichero contiene los datos que nos interesan.

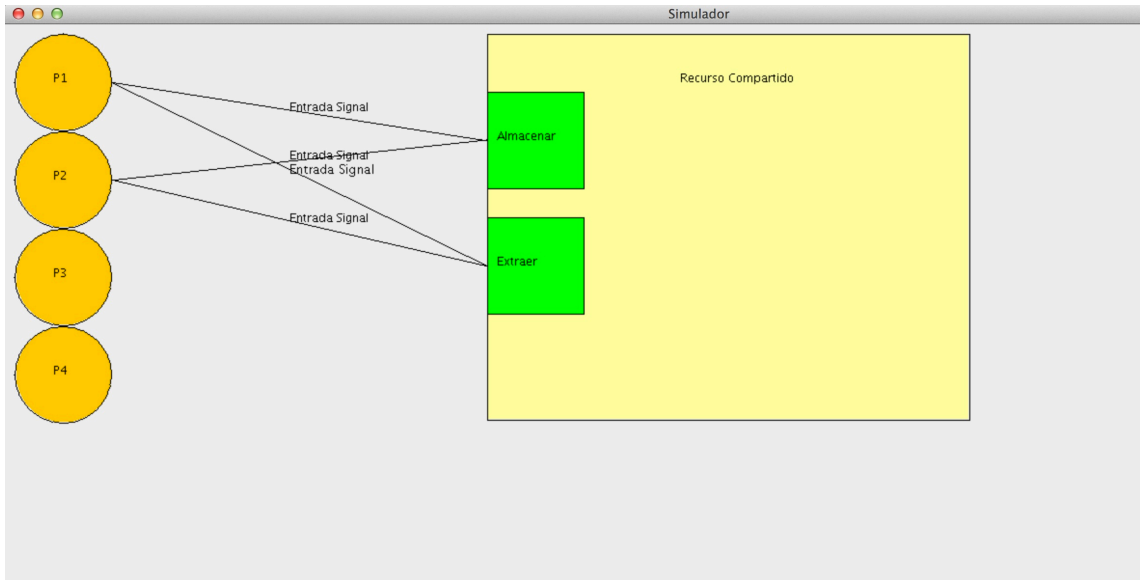


Una vez realizados los chequeos anteriores, se ejecuta el simulador de la siguiente manera.

1. Invocación del ejecutable *Simulador.jar* .



2. Imagen del simulador



6. CONCLUSIONES.

Este proyecto parte de una necesidad que se nos ha presentado a lo largo del estudio de la asignatura "Concurrencia". Así lo hemos valorado mi tutor de TFG y yo, por eso hemos pensado e la realización de una monitorización de la ejecución de los programas concurrentes, de modo que se facilitara el estudio a mis compañeros, así como la enseñanza de mi tutor y sus compañeros de departamento.

Partimos de un punto ya conocido previamente, como son las librerías de D. Julio Mariño y D. Ángel Herranz y la Univ. Kent. A esto le hemos añadido la forma en que mi tutor ha explicado a lo largo del curso esta asignatura, basándose en los formalismos gráficos, que han facilitado enormemente su estudio.

Al principio orienté el proyecto hacia la instrumentalización del código, para mostrar únicamente los datos que resultaran interesantes, y así monitorizar el código de la clase Monitor. Java, pero esto resultó inviable porque las herramientas de las que dispone el lenguaje para instrumentalizar no cubrían los requisitos que nos habíamos impuesto, y por lo tanto no podía recoger la información que considerábamos interesante.

De modo que decidimos modificar la dirección del trabajo y reconducirlo, modificando el código de la clase para que pudiera generar los eventos según nuestro interés.

Finalmente, hemos conseguido los datos que buscábamos y además hemos abierto varias posibilidades para continuar la investigación para el futuro de otros tipos de paradigma de concurrencia, como podría ser la visualización de código JCSP o semáforos.

Creo que estos caminos que quedan abiertos pueden ser un modo de trabajar ampliamente en esta área y que se pueden conseguir conclusiones importantes.

Personalmente, considero que este proyecto me ha servido de gran ayuda no sólo de cara a profundizar en el estudio de esta asignatura, sino a introducirme en el área de investigación. Hasta tal punto me ha iniciado en esta área, que no descarto seguir con proyectos similares en un futuro, si mi situación laboral me lo permite.

Me gustaría agradecer a D. Julio Mariño, tutor de este TFG, por su colaboración, orientación y apoyo a lo largo de este semestre, en el que hemos tenido varias reuniones y en las que siempre ha estado apoyándome, incluso cuando, por los resultados obtenidos, pensé que el proyecto había fracasado. Con la continuidad del esfuerzo hemos conseguido cubrir los objetivos de este TFG.

7. BIBLIOGRAFIA

Para la realización de este Trabajo Fin de Carrera ha sido necesario la lectura de manuales de Java, así como material de apoyo de la asignatura "Concurrencia". Todo ello proporcionado en las siguientes referencias.

[1] J.Mariño / A.Herranz. 2014. *Concurrencia.*[Online]. Disponible en: http://babel.ls.fi.upm.es/teaching/concurrencia/

[2] G.R. Andrews, F.B. Schneider. (1983). *Concepts and Notations for Concurrent Programming*. Disponible en: http://babel.ls.fi.upm.es/teaching/concurrencia/material/concepts_and_notations.pdf

[3] *The Java ™ Tutorials > Essential Classes, Lessons on Concurrency.* [Online] Disponible en: http://docs.oracle.com/javase/tutorial/essential/concurrency/

[4] *Java ™ Platform Overview*. [Online]. Disponible en: http://docs.oracle.com/javase/7/docs/technotes/guides/

[5] *Abstract Window Toolkit (AWT).* [Online]. Disponible en: http://docs.oracle.com/javase/7/docs/technotes/guides/awt/

[6] *Swing (Java ™ Foundation Classes)*. [Online]. Disponible en: http://docs.oracle.com/javase/7/docs/technotes/guides/swing/

8. ANEXOS

8.1. MANUAL DE USO

El presente documento muestra los pasos a seguir para que el simulador de programas concurrentes en java funcione se tiene que dar los siguientes requisitos:

1. El Recurso Compartido tiene que extender a la clase Monitor.java.

2. El Recurso Compartido tiene que implementar el interfaz Transparente.java. Esto significa que el recurso compartido tiene que ser codificado el método showMe().

   En dicho método, hay que crear una estructura con la clase contenedora Par.java, en la cual se introduce primero el par, operación, nombre de la operación en el recurso que se ha ejecutado en el momento del evento.

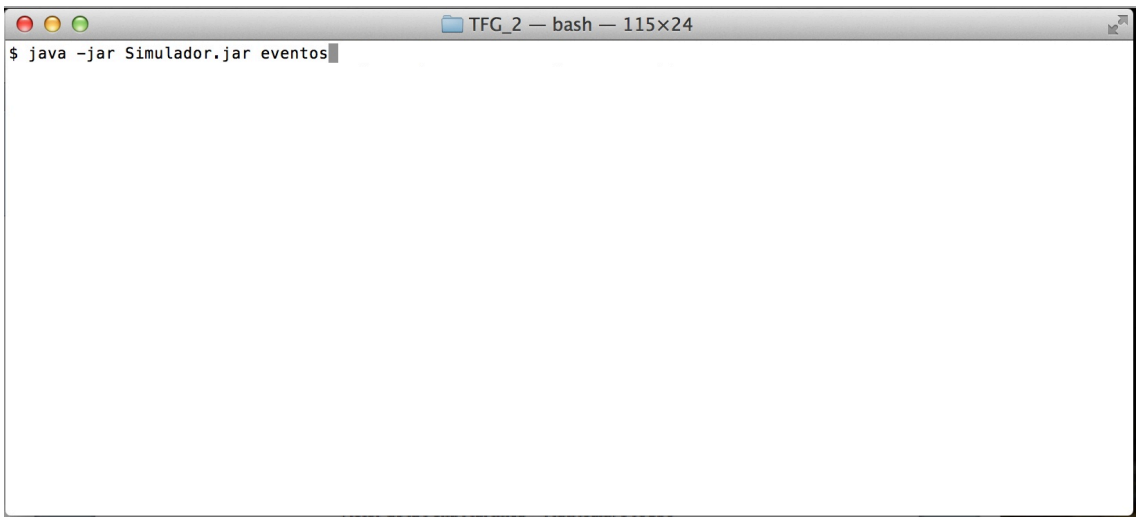
3. La clase main del problema tiene que dar la posibilidad de capturar el nombre y ruta, en caso de no incluirlo se creara un fichero llamado "eventos", en la carpeta donde este el código del problema.

Tras comprobar que se cumplen todos los requisitos anteriores. Los pasos para la ejecución del simulador es por consola:

1. `cd  <ruta donde se halla el fichero ejecutable Simulador.jar>`
2. `java –jar Simulador.jar [<Ruta> y Nombre del fichero]*.`

* Nota: Sí el fichero esta en la ruta donde esta el fichero ejecutable vale con el nombre del fichero.

Ejemplo de llamada:



8.2. GESTOR DE EVENTOS Y CAPTURA DE DICHOS EVENTOS

Monitor.java

```
package cclib1;
/**
 *
 * @author Víctor de la Peña Martinez
 *
 * Este código esta sujeto a las leyes de proteccion intelectual
Copyleft
 *
 */
import java.io.IOException;
import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.Date;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;
import java.util.concurrent.locks.Condition;
/**
 * Monitors.
 */
public class Monitor implements Transparente {
    /**
     * First barrier to mutex. This is also the barrier where signalled
     * threads are moved to.
     */
    private Lock mutex;

    /**
     * Second barrier to mutex. This is the barrier where threads
     * entering the monitor get blocked if some signalling is halfway.
     */
    private Condition purgatory;

    /**
     * Number of threads that were moved to the purgatory and have not
     * reached heaven yet (they may have been signalled).
     */
    private int inPurgatory = 0;

    /**
     * Number of signalled threads which have not completed monitor
     * re-entrance, not counting those signalled in purgatory.
     */
    private int pendingSignals = 0;

    /**
     * In order to force more arbitrary interleavings on the use of
     * Monitors, each call to await will introduce a random sleep
     * time. This attribute indicate the mean time of the sleep. If 0,
     * no sleep will be executed and the interleaving will be the
     * "natural" one.
```

```
     */
    private int meanSleepTimeAfterAwait_ms = 0;

    /**
     * Random number generator for sleeping time after await.
     */
    private java.util.Random random = new java.util.Random(1);

    /**
     * Monitor constructor. No additional initialization required.
     */
    private EventManager eventManager;
    public Monitor(EventManager eventManager) {
        this.setEventManager(eventManager);
        mutex = new ReentrantLock();
        purgatory = mutex.newCondition();
    }

    public Lock getMutex() {
      return mutex;
}


/**
     * If the parameter is greater than 0 more arbitrary interleavings
     * will be introduced.
     */
    public void setMeanSleepAfterAwait(int ms) {
       meanSleepTimeAfterAwait_ms = ms < 0 ? 0 : ms;
    }

    /**
     * Enters the monitor.
     */
    public void enter() {
       // First barrier
       mutex.lock();
       // The thread will go to purgatory unless no older blocked
       // thread exists that should be given access.
       if (pendingSignals > 0 || inPurgatory > 0) {
          inPurgatory++;
          try { purgatory.await(); }
          catch (InterruptedException e) {
             // TODO: manage this exception.
          }
          inPurgatory--;
       }
    }

    /**
     * Leaves the monitor.
     */
    public void leave() {
       // TODO: test the method is not being invoked outside of the
monitor.
```

```
      // Signal a thread in the second barrier if no thread in a
      // condition is pendingSignals.
      if (pendingSignals == 0 && inPurgatory > 0) {
         purgatory.signal();
      }
      mutex.unlock();
   }

   /**
    * Returns a new condition associated to this monitor.
    */
   public Cond newCond(EventManager eventManager) {
      return new Cond(eventManager, this);
   }

   /**
    * Conditions.
    */
   public class Cond{
      /**
       * Implementation of the queue.
       */
      private Condition condition;

      /**
       * Number of threads blocked in this condition.
       */
      private int waiting;
      private EventManager eventManager;
      private Monitor myMonitor;
      /**
       * Constructor.
       */
      public Cond(EventManager eventManager,Monitor m) {
         condition = mutex.newCondition();
         waiting = 0;
         this.myMonitor = m;
         this.eventManager = eventManager;
      }

      /**
       * The thread that invokes this method will block until
       * signalled.
     * @throws IOException
       */
      public void await() throws IOException {
         // TODO: test the method is not being invoked outside of the
monitor.
         waiting++;
         EventoImpl ev = new EventoImpl();
         StackTraceElement [] st =
Thread.currentThread().getStackTrace();
         Date utilDate = new Date();
         long timeNow = utilDate.getTime();
```

```
         Timestamp time = new Timestamp(timeNow);
         int proceso = st[2].hashCode();
         ArrayList<Par> list = myMonitor.showMe();
         ev.Eventos(proceso, "Entrada Await", list, time);
         eventManager.eventCapture(ev);
         // Before blocking we need to signal threads in the second
         // barrier since they are not automatically unblocked
         if (pendingSignals == 0 && inPurgatory > 0 ) {
            purgatory.signal();
         }
         try { condition.await(); }
         catch (InterruptedException e) {
            // TODO: manage this exception.
         }
         pendingSignals--;
         if (meanSleepTimeAfterAwait_ms > 0) {
            try {
               Thread.sleep(random.nextInt(2 *
meanSleepTimeAfterAwait_ms));
            } catch (InterruptedException e) {
            }
         }
         ev = new EventoImpl();
         //st = Thread.currentThread().getStackTrace();
         utilDate = new Date();
         timeNow = utilDate.getTime();
         time = new Timestamp(timeNow);
         proceso = st[2].hashCode();
         list = myMonitor.showMe();
         ev.Eventos(proceso, "Salida Await", list, time);
         eventManager.eventCapture(ev);
      }

      /**
       * Signal (and continue) the first thread blocked in the
       * condition.
     * @throws IOException
       */
      public void signal() throws IOException {
         // TODO: test the method is not being invoked outside of the
monitor.
       EventoImpl ev = new EventoImpl();
          StackTraceElement [] st =
Thread.currentThread().getStackTrace();
          Date utilDate = new Date();
          long timeNow = utilDate.getTime();
          Timestamp time = new Timestamp(timeNow);
          int proceso = st[2].hashCode();
          ArrayList<Par> list = myMonitor.showMe();
          ev.Eventos(proceso, "Entrada Signal", list, time);
          eventManager.eventCapture(ev);
         if (waiting > 0) {
            if (pendingSignals > 1) {
              throw new Error("Just one pending signal is allowed");
            }
```

```
            pendingSignals++;
            waiting--;
            // Moving a thread to the first barrier
            condition.signal();
         }
         ev = new EventoImpl();
         //st = Thread.currentThread().getStackTrace();
         utilDate = new Date();
         timeNow = utilDate.getTime();
         time = new Timestamp(timeNow);
         proceso = st[2].hashCode();
         list = myMonitor.showMe();
         ev.Eventos(proceso, "Salida Signal", list, time);
         eventManager.eventCapture(ev);
      }

      /**
       * Signal (and continue) all the threads blocked in the
       * condition.
       *
       * WARNING: We recommend not to use signalAll since the current
       * internal state of a resource is not guarantee for all
       * signalled threads (except for the first one).
       */
      @SuppressWarnings("unused")
      private void signalAll() {
       // TODO: test the method is not being invoked outside of the
monitor.
       EventoImpl ev = new EventoImpl();
       StackTraceElement [] st =
Thread.currentThread().getStackTrace();
       Date utilDate = new Date();
       long timeNow = utilDate.getTime();
       Timestamp time = new Timestamp(timeNow);
       int proceso = st[2].hashCode();
       ArrayList<Par> list = myMonitor.showMe();
       ev.Eventos(proceso, "Entrada signalAll", list, time);
       try {
         eventManager.eventCapture(ev);
       } catch (IOException e1) {
         // TODO Auto-generated catch block
         e1.printStackTrace();
       }
       pendingSignals += waiting;
       waiting = 0;
       // Moving all the threads to the first barrier
       condition.signalAll();
       ev = new EventoImpl();
       utilDate = new Date();
       timeNow = utilDate.getTime();
       time = new Timestamp(timeNow);
       proceso = st[2].hashCode();
       list = myMonitor.showMe();
       ev.Eventos(proceso, "Entrada await", list, time);
       try {
```

```
        eventManager.eventCapture(ev);
       } catch (IOException e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
       }
      }

      /**
       * Number of threads waiting in this Cond.
       */
      public int waiting() {
         // TODO: test the method is not being invoked outside of the
monitor.
         return waiting;
      }
   }

@Override
public ArrayList<Par> showMe() {
     // TODO Auto-generated method stub
     return null;
}

public EventManager getEventManager() {
     return eventManager;
}

public void setEventManager(EventManager eventManager) {
     this.eventManager = eventManager;
}
}
```

Evento.java

```
package cclib1;
/**
 *
 * @author Víctor de la Peña Martinez
 *
 * Este código esta sujeto a las leyes de proteccion intelectual
Copyleft
 *
 */
import java.sql.Timestamp;
import java.util.ArrayList;



public interface Evento {
     public void setList(ArrayList<Par> list);
     public StackTraceElement [] getStack();

     public void setStack(StackTraceElement [] stack) ;

     public Object getObjeto();

     public void setObjeto(Object objeto);

     public Timestamp getTimeEvent() ;

     public void setTimeEvent(Timestamp timeEvent) ;

     public ArrayList<Par> getArrayListPar();

     public int getNombreProceso();

     public void setNombreProceso(int nombreProceso);

     public String getOperacion();

     public void setOperacion(String operacion);
    //public void Evento(StackTraceElement [] st, Object
ob,ArrayList<Par> list ,Timestamp time);

    public void Eventos(int nombreProceso,String
operacion,ArrayList<Par> list ,Timestamp time);
}
```

EventoImpl.java

```
package cclib1;
/**
 *
 * @author Víctor de la Peña Martinez
 *
 * Este código esta sujeto a las leyes de proteccion intelectual
Copyleft
 *
 */
import java.io.Serializable;
import java.sql.Timestamp;
import java.util.ArrayList;

@SuppressWarnings("serial")
public class EventoImpl implements Evento, Serializable{
     private StackTraceElement [] stack;
     private Object objeto;
     private Timestamp timeEvent;
     private ArrayList<Par> list;
     private int nombreProceso;
     private String operacion;

     public void setList(ArrayList<Par> list) {
       this.list = list;
     }

     public StackTraceElement [] getStack() {
       return stack;
     }

     public void setStack(StackTraceElement [] stack) {
       this.stack = stack;
     }

     public Object getObjeto() {
       return objeto;
     }

     public void setObjeto(Object objeto) {
       this.objeto = objeto;
     }

     public Timestamp getTimeEvent() {
       return timeEvent;
     }

     public void setTimeEvent(Timestamp timeEvent) {
       this.timeEvent = timeEvent;
     }
     /*@Override
     // Informacion desde que thread (Stacktrace) y el objeto que ha
llamado y timestamp
     public void Evento(StackTraceElement [] st, Object ob,
```

```
ArrayList<Par> list ,Timestamp time) {
      // TODO Auto-generated method stub
      this.setObjeto(ob);
      this.setStack(st);
      this.setTimeEvent(time);
      this.setList(list);
    }*/

    @Override
    public ArrayList<Par> getArrayListPar() {
      // TODO Auto-generated method stub
      return list;
    }


    @Override
    public void Eventos(int nombreProceso, String operacion,
ArrayList<Par> list, Timestamp time) {
      // TODO Auto-generated method stub
      this.setNombreProceso(nombreProceso);
      this.setOperacion(operacion);
      this.setTimeEvent(time);
      this.setList(list);
    }

    public int getNombreProceso() {
      return nombreProceso;
    }

    public void setNombreProceso(int nombreProceso) {
      this.nombreProceso = nombreProceso;
    }

    public String getOperacion() {
      return operacion;
    }

    public void setOperacion(String operacion) {
      this.operacion = operacion;
    }
}
```

Transparente.java

```
package cclib1;
/**
 *
 * @author Víctor de la Peña Martinez
 *
 * Este código esta sujeto a las leyes de proteccion intelectual
Copyleft
 *
 */
import java.util.ArrayList;

public interface Transparente {

     public ArrayList<Par> showMe();
}
```

Par.java

```
package cclib1;
/**
 *
 * @author Víctor de la Peña Martinez
 *
 * Este código esta sujeto a las leyes de proteccion intelectual
Copyleft
 *
 */
public class Par {
private String name;
public String getName() {
     return name;
}
public void setName(String name) {
     this.name = name;
}
public Par(String name, String operacion) {
     super();
     this.name = name;
     this.value = operacion;
}
public String getValue() {
     return value;
}
public void setValue(String value) {
     this.value = value;
}
private String value;
}
```

EventManager.java

```
package cclib1;
/**
 *
 * @author Víctor de la Peña Martinez
 *
 * Este código esta sujeto a las leyes de proteccion intelectual
Copyleft
 *
 */
import java.io.*;
import java.util.*;

public class EventManager {//implements Serializable{

    /**
     *
     */
    private LinkedList<Evento> listEvent;// = new LinkedList<>();
    private int index;
    private  FileWriter fw;
    private  PrintWriter br;
    public LinkedList<Evento> getListEvent() {
      return listEvent;
    }
    public void setListEvent(LinkedList<Evento> listEvent) {
      this.listEvent = listEvent;
      index = 0;
    }
    public EventManager() {
      //super();
      //Abrir fichero
           try
           {
              fw = new FileWriter("/Users/victor/Desktop/prueba22");
              br = new PrintWriter(fw);
              br.close();
              fw.close();
           }
           catch(Exception e)
           {
               e.printStackTrace();
               System.out.println("Error");
           }
         this.listEvent = new LinkedList<>();
         index = 0;
    }
    private void print() throws IOException{
      Evento ev = listEvent.getLast();
        fw = new FileWriter("/Users/victor/Desktop/prueba22",true);
        br = new PrintWriter(fw);
      br.write("==============="+index+"===============");
      br.println();
```

```
        br.write("nombreProceso: ");
        br.println();
        br.write(""+ev.getNombreProceso());
        br.println();
        br.write("Operacion: ");
        br.println();
        br.write(ev.getOperacion());
        br.println();
        ArrayList<Par> listaAux = ev.getArrayListPar();
        Iterator <Par> it = listaAux.iterator();
        br.write("Variables Recurso Compartido:");
        br.println();
        while (it.hasNext())
        {
              Par par = it.next();
              br.write(par.getName()+": "+par.getValue());
              br.println();
        }
        br.write("+++++++++++++++");
        br.println();
        br.write("TimeStamp: ");
        br.println();
        br.write(ev.getTimeEvent().toString());
        br.println();
        br.write("================"+index+"================");
        br.println();
        index++;
       br.close();
       fw.close();
      }
      public void eventCapture(Evento ev) throws IOException{
        listEvent.addFirst(ev);
        print();
        listEvent.removeFirst();
      }

}
```

8.3. VISUALIZADOR DE EVENTOS

Values.java

```
package cclib1;
/**
 *
 * @author Víctor de la Peña Martinez
 *
 * Este código esta sujeto a las leyes de proteccion intelectual
Copyleft
 *
 */
public class Values {
     private String string;
     private String figura;
     private int centroX;
     private int centroY;
     private int ancho;
     private int alto;
     public Values(int x, int y, int ancho, int alto, String figura,
String string) {
       super();
       this.centroX = x;
       this.centroY = y;
       this.ancho = ancho;
       this.alto = alto;
       this.figura = figura;
       this.string = string;
     }
     public int getCentroX() {
       return centroX;
     }
     public int getCentroY() {
       return centroY;
     }
     public int getAlto() {
       return alto;
     }
     public int getAncho() {
       return ancho;
     }
     public String getFigura() {
       return figura;
     }
     public String getString() {
       return string;
     }
}
```

Simulate.java

```
package model;
/**
 *
 * @author Víctor de la Peña Martinez
 *
 * Este código esta sujeto a las leyes de proteccion intelectual
Copyleft
 *
 */
import java.util.ArrayList;
import java.util.Iterator;

import cclib1.Evento;
import cclib1.Par;
import cclib1.Values;
public class Simulate {
    private ArrayList<Evento> listaFichero;
    private ArrayList<Integer> numProcesos;

    public Simulate(ArrayList<Evento> listaFichero,ArrayList<Integer>
numProcesos) {
      super();
      this.listaFichero = listaFichero;
      this.numProcesos = numProcesos;
    }
    public ArrayList<Evento> getListaFichero() {
      return listaFichero;
    }
    public void setListaFichero(ArrayList<Evento> listaFichero) {
      this.listaFichero = listaFichero;
    }
    public ArrayList<Integer> getNumProcesos() {
      return numProcesos;
    }
    public void setNumProcesos(ArrayList<Integer> numProcesos) {
      this.numProcesos = numProcesos;
    }
    public void simulate() {

      ArrayList<Values> listProcess = createProcess();
      Values valorRecurso = new Values(500, 10, 500, 400, "Recurso
Compartido", "Recurso Compartido");
      Values valorOp1 = new Values(500, 70, 100, 100, "Operación",
"Almacenar");
      Values valorOp2 = new Values(500, 200, 100, 100, "Operación",
"Extraer");
      ArrayList<Values> listEvent = createEvents(listProcess,
valorOp1, valorOp2);
      listProcess.add(valorRecurso);
      listProcess.add(valorOp1);
      listProcess.add(valorOp2);
      Visual v = new Visual(listProcess, listEvent);
      long t=System.currentTimeMillis();
```

```
      int retardo = 800;
      Iterator <Values>itE = listEvent.iterator();
        while (itE.hasNext()) {
            v.mover();
            try{
                t+=retardo;
                Thread.sleep(Math.max(0, t-
System.currentTimeMillis()));
            }catch(InterruptedException ex){
                break;
            }
        }

    }
    private ArrayList<Values> createEvents(ArrayList<Values> listP,
Values operacion1, Values operacion2) {
      ArrayList <Values> listE = new ArrayList<Values>();
      Iterator<Evento> it = listaFichero.iterator();
      while(it.hasNext())
      {
            Evento ev = it.next();
            Iterator<Values> itP = listP.iterator();
            boolean bool = true;
            while ((bool)&&(itP.hasNext()))
            {
                  Values v = itP.next();
                  ArrayList<Par> listaPar = ev.getArrayListPar();
                  Par par = buscaOperacion(listaPar);
                  if ((ev.getNombreProceso() ==
Integer.parseInt(v.getString())))

    if(par.getValue().compareTo(operacion1.getString()) == 0)
                        {
                              Values val = new
Values(v.getCentroX()+v.getAncho(), (v.getCentroY()+(v.getAlto()/2)),
(operacion1.getCentroX()),
(operacion1.getCentroY()+(operacion1.getAlto()/2)), "Evento",
ev.getOperacion());
                              listE.add(val);
                              bool = false;
                        }

    if(par.getValue().compareTo(operacion2.getString()) == 0)
                        {

                              Values val = new
Values(v.getCentroX()+v.getAncho(), (v.getCentroY()+(v.getAlto()/2)),
(operacion2.getCentroX()),
(operacion2.getCentroY()+(operacion2.getAlto()/2)), "Evento",
ev.getOperacion());
                              listE.add(val);
                              bool = false;
                        }

                  }
```

```
            }
        }

        return listE;
    }
    private Par buscaOperacion(ArrayList<Par> listaPar) {
        Par par = new Par("", "");
        Iterator<Par>itPar = listaPar.iterator();
        boolean bool = false;
        while ((itPar.hasNext())&&(!bool))
        {
            Par aux = itPar.next();
            if(aux.getName().compareTo("Operación:") == 0)
            {
                par = aux;
                bool = true;
            }
        }
        return par;
    }
    private ArrayList<Values> createProcess() {
        ArrayList <Values> listP = new ArrayList<Values>();
        Iterator <Integer> it = numProcesos.iterator();
        int index = 0;
        while(it.hasNext())
        {
            Values v = new Values(0+10, ((100*index)+(10+index)), 100,
100, "Proceso" , it.next().toString());
            listP.add(v);
            index++;
        }
        return listP;
    }

}
```

Visual.java

```
package model;
/**
 *
 * @author Víctor de la Peña Martinez
 *
 * Este código esta sujeto a las leyes de protección intelectual
Copyleft
 *
 */
import javax.swing.*;
import javax.swing.border.EmptyBorder;

import java.awt.*;
import java.awt.image.ImageObserver;
import java.text.AttributedCharacterIterator;
import java.util.ArrayList;
import java.util.Iterator;

import cclib1.*;

import javax.swing.GroupLayout.Alignment;
public class Visual extends JPanel{

     private JPanel panel;
     private ArrayList<Values>val;
     private ArrayList<Values>event;
     private JFrame frame;
     private Iterator<Values> itE;
     private ArrayList<Values> eventosAuxiliar;
     private Thread th;
     private Values valor;
     private int retardo;
     /**
      *
      * Create the frame.
      */
     public Visual(ArrayList<Values> val, ArrayList<Values> eventos) {
       setBackground(new Color(238, 238, 238));
       frame = new JFrame("Simulador");
          panel = this;
          frame.setSize(2000, 1000);
          frame.getContentPane().add(panel);
          frame.getContentPane().setLayout(new GridLayout(2, 2));
          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       frame.setVisible(true);
          panel.setSize(2000, 1000);
          frame.getContentPane().add(panel);
          frame.getContentPane().setLayout(new GridLayout(2, 2));
          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       frame.setVisible(true);
       this.event = eventos;
       this.val = val;
```

```
      valor = new Values(0, 0, 0, 0, "", "");
      retardo = 50;
      itE = event.iterator();
      eventosAuxiliar = new ArrayList<Values>();
    }
    public void paintComponent(Graphics g) {
       drawFigures(g);
    }

    public void drawFigures (Graphics g){
      Iterator<Values> it = val.iterator();
      while(it.hasNext())
      {
            Values coord = it.next();
            if(coord.getFigura().compareTo("Proceso") == 0)
            {
                  Color cAnt = Color.black;
                  Color c = Color.orange;
                  g.setColor(c);
                  g.fillOval(coord.getCentroX()+1, coord.getCentroY(),
coord.getAncho()-1, coord.getAlto()-1);
                  g.setColor(cAnt);
                  g.drawOval(coord.getCentroX(), coord.getCentroY(),
coord.getAncho(), coord.getAlto());
                  g.drawString(coord.getString(),
coord.getCentroX()+10, coord.getCentroY()+50);
            }
            if (coord.getFigura().compareTo("Recurso Compartido") ==
0)
            {
                  Color cAnt = Color.black;
                  Color c = new Color(000-150-200);
                  g.setColor(c);
                  g.fillRect(coord.getCentroX()+1,
coord.getCentroY()+1, coord.getAncho()-2, coord.getAlto()-2);
                  g.setColor(cAnt);
                  g.drawRect(coord.getCentroX(), coord.getCentroY(),
coord.getAncho(), coord.getAlto());

    g.drawString(coord.getString(),coord.getCentroX()+200,coord.getCen
troY()+50);
            }
            if (coord.getFigura().compareTo("Operación") == 0)
            {
                  Color cAnt = Color.black;
                  Color c = Color.GREEN;
                  g.setColor(c);
                  g.fillRect(coord.getCentroX()+1,
coord.getCentroY()+1, coord.getAncho()-1, coord.getAlto()-1);
                  g.setColor(cAnt);
                  g.drawRect(coord.getCentroX(), coord.getCentroY(),
coord.getAncho(), coord.getAlto());

    g.drawString(coord.getString(),coord.getCentroX()+10,coord.getCent
roY()+50);
```

```
            }
      }
    }

    public void paint(Graphics g) {
      drawFigures(g);
      int alto = 0;
      int ancho = 0;
      if (Esta())
      {
            Values vAux = darValor();
            Color cAux = new Color(238, 238, 238);
            g.setColor(cAux);
            g.drawLine(vAux.getCentroX(), vAux.getCentroY(),
vAux.getAncho(), vAux.getAlto());
            ancho = (vAux.getCentroX()+vAux.getAncho())/2;
            alto = (vAux.getCentroY()+vAux.getAlto())/2;
            g.drawString(vAux.getString(),ancho-10,alto);
            Color c = Color.black;
            g.setColor(c);
      }
            eventosAuxiliar.add(valor);
            g.drawLine(valor.getCentroX(), valor.getCentroY(),
valor.getAncho(), valor.getAlto());
            ancho = (valor.getCentroX()+valor.getAncho())/2;
            alto = (valor.getCentroY()+valor.getAlto())/2;
            g.drawString(valor.getString(),ancho-10,alto);

    }
    private Values darValor() {
      // TODO Auto-generated method stub
      Values vAux = new Values(0, 0, 0, 0, "", "");
      Iterator <Values> itAux = eventosAuxiliar.iterator();
      boolean esta = false;
      while ((itAux.hasNext())&&(!esta))
      {
            vAux = itAux.next();
            if ((vAux.getCentroX() ==
valor.getCentroX())&&(vAux.getCentroY() == valor.getCentroY())
                        &&(vAux.getAlto() ==
valor.getAlto())&&(vAux.getAncho() == valor.getAncho())&&

    (vAux.getString().compareTo(valor.getString())!= 0))
            {
                  eventosAuxiliar.remove(vAux);
                  esta = true;
            }
      }
      return vAux;
    }
    private boolean Esta() {
      // TODO Auto-generated method stub
      boolean esta = false;
      Iterator <Values> itAux = eventosAuxiliar.iterator();
      while ((itAux.hasNext())&&(!esta))
```

```
      {
            Values vAux = itAux.next();
            if ((vAux.getCentroX() ==
valor.getCentroX())&&(vAux.getCentroY() == valor.getCentroY())
                        &&(vAux.getAlto() ==
valor.getAlto())&&(vAux.getAncho() == valor.getAncho())&&

    (vAux.getString().compareTo(valor.getString())!= 0))
            {
                  esta = true;
            }
      }
      return esta;
    }
    public void simule(Graphics g) {
      // TODO Auto-generated method stub
      //Iterator<Values> it = event.iterator();
      if (itE.hasNext())
      {
            Values val = itE.next();
            g.drawLine(valor.getCentroX(), valor.getCentroY(),
valor.getAncho(), valor.getAlto());

    g.drawString(valor.getString(),valor.getCentroX()+100,valor.getCen
troY()+5);
            int ancho = (valor.getCentroX()+valor.getAncho())/2;
            int alto = (valor.getCentroY()+valor.getAlto())/2;
            g.drawString(valor.getString(),ancho-10,alto);

      }
    }
    public void mover() {
      // TODO Auto-generated method stub
      if (itE.hasNext())
      {
            valor = itE.next();
            repaint();
      }
    }
}
```

MainModel.java

```
package model;
/**
 *
 * @author Víctor de la Peña Martinez
 *
 * Este código esta sujeto a las leyes de proteccion intelectual
Copyleft
 *
 */
import java.io.*;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.*;

import cclib1.*;

public class MainModel {
    /*
     * metodo para leer el fichero.
     */
    private static ArrayList<Evento> leerFichero(String nombreFichero)
{
      ArrayList<Evento> lista = new ArrayList<Evento>();
      File archivo;
      FileReader punteroLector;
      BufferedReader buffer;
      try
      {
            archivo = new File (nombreFichero);
            punteroLector = new FileReader(archivo);
          buffer = new BufferedReader(punteroLector);
            String linea;
            while ((linea = buffer.readLine())!= null)
            {
                  Evento ev = new EventoImpl();
                  if (!linea.contains("========"))
                  {
                        if (linea.contains("nombreProceso: "))
                        {


    ev.setNombreProceso(Integer.parseInt(buffer.readLine()));
                        }
                        linea = buffer.readLine();
                        if (linea.contains("Operacion: "))
                        {
                              ev.setOperacion(buffer.readLine());
                        }
                        linea = buffer.readLine();
                        if (linea.contains("Variables Recurso
Compartido:"))
                        {
                              linea = buffer.readLine();
```

```
                              ArrayList<Par> listaVariables = new
ArrayList<Par>();
                              String delim =" ";


    while((linea.compareTo("+++++++++++++++"))!= 0)
                              {
                                    String [] aux = linea.split(delim);
                                    Par p = new Par(aux[0], aux[1]);
                                    listaVariables.add(p);
                                    linea = buffer.readLine();
                              }
                              ev.setList(listaVariables);
                        }
                        linea = buffer.readLine();
                        if (linea.contains("TimeStamp: "))
                        {
                              SimpleDateFormat formatoFecha = new
SimpleDateFormat("yyyy-mm-dd hh:mm:ss.SSS");
                              Date fecha =
formatoFecha.parse(buffer.readLine());
                              ev.setTimeEvent(new
java.sql.Timestamp(fecha.getTime()));
                        }
                        lista.add(ev);
                  }
            }
            buffer.close();
      }
      catch (FileNotFoundException e)
      {
            e.printStackTrace();
      }
      catch (IOException e)
      {
            e.printStackTrace();
      }
      catch (ParseException e)
      {
            e.printStackTrace();
      }
      return lista;
    }
    /*
     * Funcion que devuelve numero de procesos distintos
     */
    private static ArrayList<Integer> numeroProcesos(ArrayList<Evento>
listaFichero) {
      ArrayList<Integer> numeroProcesos = new ArrayList<Integer>();
      Iterator <Evento>it = listaFichero.iterator();
      while (it.hasNext())
      {
            Evento ev = it.next();
            if (!numeroProcesos.contains(ev.getNombreProceso()))
            {
```

```
                    numeroProcesos.add(ev.getNombreProceso());
              }
        }
        return numeroProcesos;
      }

      private static void print(ArrayList<Evento> listaFichero) {
        Iterator<Evento> it = listaFichero.iterator();
        int index = 0;
        while (it.hasNext())
        {
              Evento ev = it.next();
              System.out.println("Numero de evento: "+index);
              System.out.println("Nombre: "+ev.getNombreProceso());
              System.out.println("Operación: "+ev.getOperacion());
              System.out.println("ListPar: "+ev.getArrayListPar());
              System.out.println("tiempo:
"+ev.getTimeEvent().toString());
              System.out.println("=========================");
              index++;
        }

      }
      public static void main(String[] args) {
        String nombreFichero =
"/Users/victor/Desktop/prueba22";//args[1];
        ArrayList<Evento>listaFichero=leerFichero(nombreFichero);
        System.out.println("Lista fichero: "+listaFichero.size());
        //print(listaFichero);
        System.out.println("Lista fichero: "+listaFichero.size());
        ArrayList<Integer> numProcesos= numeroProcesos(listaFichero);
        System.out.println("numero de Procesos leidos:
"+numProcesos.size());
        Simulate sim = new Simulate(listaFichero,numProcesos);
        sim.simulate();
      }
```

8.4. CÓDIGO EJEMPLO

Escritor.java

```
package LectoresEscritores;
/**
 *
 * @author Víctor de la Peña Martinez
 *
 * Este código esta sujeto a las leyes de proteccion intelectual
Copyleft
 *
 */
public class Escritor extends Thread{
     private RecursoCompartido almac;
     private int numMaxCaracteres;
     //private char [] cadena;
     public Escritor(int N, RecursoCompartido almac) {
       // TODO Auto-generated constructor stub
       this.setNumMaxCaracteres(N);
       this.setAlmac(almac);
       //cadena = new char[numMaxCaracteres];
     }
     public int getNumMaxCaracteres() {
       return numMaxCaracteres;
     }
     public void setNumMaxCaracteres(int numMaxCaracteres) {
       this.numMaxCaracteres = numMaxCaracteres;
     }
     public RecursoCompartido getAlmac() {
       return almac;
     }
     public void setAlmac(RecursoCompartido almac) {
       this.almac = almac;
     }
     public void run(){
       try
       {
             System.out.println("numMaxCaracteres: "+numMaxCaracteres);
             /*for (int i=0; i <= numMaxCaracteres; i++)
             {
                   cadena[i]='A';
             }*/
             almac.almacenar('A');
             //System.out.println("Va a escribir en el recurso
compartido la cadena: [A].");
             sleep(5);
             almac.almacenar('B');
             //System.out.println("Va a escribir en el recurso
compartido la cadena: [B].");
             sleep(5);
             almac.almacenar('C');
             //System.out.println("Va a escribir en el recurso
compartido la cadena: [C].");
```

```
            sleep(5);
            almac.almacenar('D');
            //System.out.println("Va a escribir en el recurso
compartido la cadena: [D].");
            sleep(5);
            //almac.almacenar('E');
            System.out.println("Va a escribir en el recurso compartido
la cadena: [E].");
            //almac.almacenar(cadena);
      } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
      }
      finally
      {

      }
    }
}
```

Lector.java

```
package LectoresEscritores;
/**
 *
 * @author Víctor de la Peña Martinez
 *
 * Este código esta sujeto a las leyes de proteccion intelectual
Copyleft
 *
 */
public class Lector extends Thread{
     private RecursoCompartido almac;
     private int numMaxCaracteres;
     //private char [] cadena;
     public Lector(int N,RecursoCompartido almac) {
       // TODO Auto-generated constructor stub
       this.setNumMaxCaracteres(N);
       this.setAlmac(almac);
       //cadena = new char[numMaxCaracteres];
     }
     public RecursoCompartido getAlmac() {
       return almac;
     }
     public void setAlmac(RecursoCompartido almac) {
       this.almac = almac;
     }
     public int getNumMaxCaracteres() {
       return numMaxCaracteres;
     }
     public void setNumMaxCaracteres(int numMaxCaracteres) {
       this.numMaxCaracteres = numMaxCaracteres;
     }
     public void run(){
       try
       {
             /*cadena = almac.extraer(numMaxCaracteres);
             System.out.println("El lector ha leido y extraido la
cadena: ["+cadena+"].")*/
             char caracter = almac.extraer(0);
             //System.out.println("Va a leer en el recurso compartido
la cadena: ["+caracter+"].");
             sleep(5);
             caracter = almac.extraer(1);
             //System.out.println("Va a leer en el recurso compartido
la cadena: ["+caracter+"].");
             sleep(5);
             caracter = almac.extraer(2);
             //System.out.println("Va a leer en el recurso compartido
la cadena: ["+caracter+"].");
             sleep(5);
             caracter = almac.extraer(3);
             //System.out.println("Va a leer en el recurso compartido
la cadena: ["+caracter+"].");
```

```
            sleep(5);
            caracter = almac.extraer(4);
            //System.out.println("Va a leer en el recurso compartido
la cadena: ["+caracter+"].");
      } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
      }
      finally
      {

      }
    }
}
```

RecursoCompartido.java

```
package LectoresEscritores;
/**
 *
 * @author Víctor de la Peña Martinez
 *
 * Este código esta sujeto a las leyes de proteccion intelectual
Copyleft
 *
 */
import java.io.IOException;
import java.util.ArrayList;
import cclib1.*;

public class RecursoCompartido extends Monitor implements
Transparente{
     private int capacidad = 6;
     private char[] almacenado = null;
     private int aExtraer = 0;
     private int aInsertar = 0;
     private int nDatos = 0;
     private String operacion;
     Cond cAlmacenar;
     Cond cExtraer;
     public RecursoCompartido(EventManager eventManager) {
       // TODO Auto-generated constructor stub
       super(eventManager);
       cAlmacenar = this.newCond(eventManager);
       cExtraer = this.newCond(eventManager);
       almacenado = new char[capacidad];
       aExtraer = 0;
       aInsertar = 0;
       nDatos = 0;
     }
     private int nDatos() {
       return nDatos;
     }

     private int nHuecos() {
       return capacidad - nDatos;
     }
     public void almacenar(char productos) {
       // TODO: implementación de código de bloqueo para
       // exclusión muytua y sincronización condicional
       // mutex.lock();
       operacion = "Almacenar";
       this.enter();
       // código de
       // sincronización para poder almacenar.
       if (nHuecos() < 0)
       {
             try {
                   //System.out.println("Hago await");
                   cAlmacenar.await();
```

```
            } catch (IOException e) {
                  // TODO Auto-generated catch block
            }
      }
      // Sección crítica
            //almacenado.add(aInsertar, productos[i]);
            almacenado[aInsertar] = productos;
            //System.out.println("Caracter a coger: "+productos);
            nDatos++;
            aInsertar++;
            aInsertar %= capacidad;
            nDatos++;
      // TODO: implementación de código de desbloqueo para
      // sincronización condicional y liberación de la exclusión mutua
      try {
            cExtraer.signal();
      } catch (IOException e) {
            // TODO Auto-generated catch block
      }
      //mutex.unlock();
      operacion = "Almacenar";
      this.leave();
}
public char extraer(int n) {
      char result = '0';

      // TODO: implementación de código de bloqueo para exclusión
      // mutua y sincronización condicional
      operacion = "Extraer";
      this.enter();
      // Sección crítica
      if (nDatos() == capacidad) {
            try {
                  //System.out.println("Hago await el lector");
                  cExtraer.await();
            } catch (IOException e) {
            }
      }
      //for (int i = 0; i < n; i++) {
            result = almacenado[aExtraer];
            almacenado[aExtraer] = '0';
            nDatos--;
            aExtraer++;
            aExtraer %= capacidad;
      //}
      try {
            cAlmacenar.signal();
      } catch (IOException e) {
      }
      operacion = "Extraer";
      this.leave();

      return result;
}
```

```
    @Override
    public ArrayList<Par> showMe() {
      // TODO Auto-generated method stub
      ArrayList<Par> list = new ArrayList<>();
      Par op = new Par ("Operación",operacion);
      Par cap = new Par ("capacidad",Integer.toString(capacidad));
      Par nData = new Par("nDatos", Integer.toString(nDatos));
      list.add(op);
      list.add(cap);
      list.add(nData);
      //System.out.println("Esta llamando a showMe de Recurso
Compartido");
      //System.out.println("Capacidad: "+capacidad);
      //System.out.println("nDatos: "+nDatos);
      return list;
    }
}
```

Main.java

```
package LectoresEscritores;
/**
 *
 * @author Víctor de la Peña Martinez
 *
 * Este código esta sujeto a las leyes de proteccion intelectual
Copyleft
 *
 */
import cclib1.EventManager;


class Main {
    public static final void main(final String[] args) throws
InterruptedException {
     int N = 6;
        // Número de productores y consumidores
        final int N_PRODS = 2;
        final int N_CONSS = 2;

        EventManager eventManager = new EventManager(args[0]);
        // Almacen compartido
        RecursoCompartido almac = new RecursoCompartido(eventManager);
        // Declaración de los arrays de productores y consumidores
        Escritor[] productores;
        Lector[] consumidores;

        // Creación de los arrays
        productores = new Escritor[N_PRODS];
        consumidores = new Lector[N_CONSS];

        // Creación de los productores
        for (int i = 0; i < N_PRODS; i++) {
            productores[i] = new Escritor(N,almac);
        }

        // Creación de los consumidores
        for (int i = 0; i < N_CONSS; i++) {
            consumidores[i] = new Lector(N, almac);
        }

        // Lanzamiento de los productores
        for (int i = 0; i < N_PRODS; i++) {
            productores[i].start();
        }
        for (int i = 0; i < N_CONSS; i++) {
            consumidores[i].start();
        }

        // Espera hasta la terminación de los procesos
        try {
            for (int i = 0; i < N_PRODS; i++) {
```

```
                productores[i].join();
            }
            for (int i = 0; i < N_CONSS; i++) {
                consumidores[i].join();
            }
        } catch (Exception ex) {
            ex.printStackTrace();
            System.exit (-1);
        }
    }
}
```

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES
	Fecha/Hora	Fri Jun 06 21:57:10 CEST 2014
	Emisor del Certificado	EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES
	Numero de Serie	630
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)